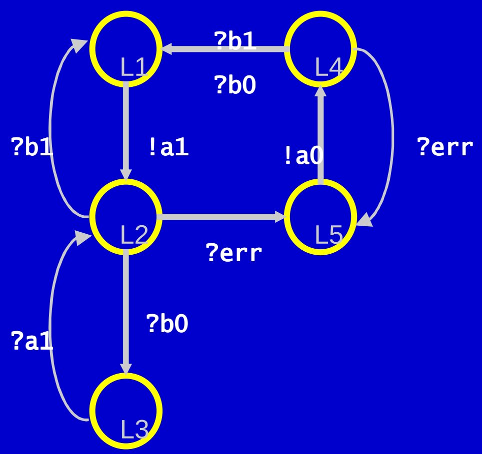
Temporal Logics

- Temporal Logics (CTL, ACTL)
- Logic patterns

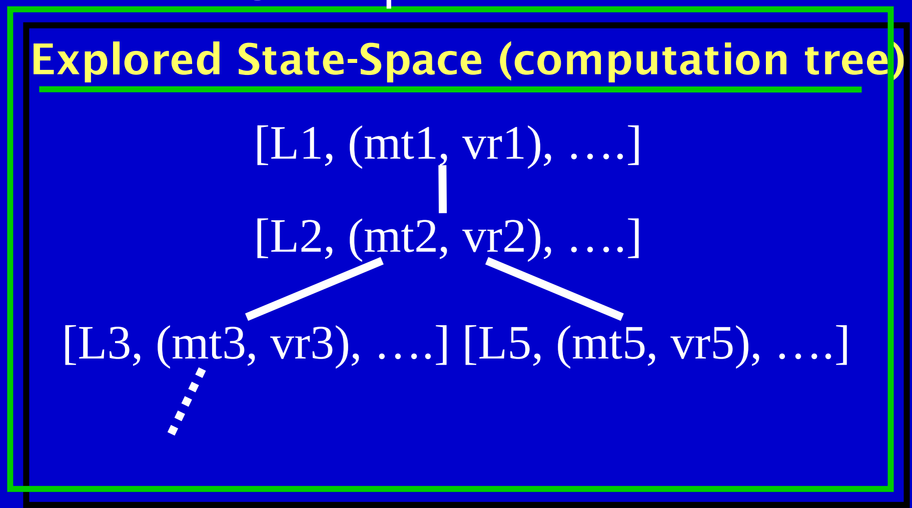
SSDE

Eric Madelaine -- vendredi 27 mars 2009

Reasoning about Executions



Conceptual View



- We want to reason about execution trees
  - tree node = snap shot of the program's state
- Reasoning consists of two layers
  - defining predicates on the program states (control points, variable values)
  - expressing temporal relationships between those predicates

Computational Tree Logic (CTL)

Clarke & Emerson (early 1980's)

Syntax

```
Φ ::=  P                                              ...primitive propositions
     | !Φ  | Φ && Φ | Φ || Φ | Φ -> Φ               ...propositional connectives
     | AG Φ | EG Φ | AF Φ | EF Φ                      ...temporal operators
     | AX Φ | EX Φ | A[Φ U Φ] | E[Φ U Φ]
```

Semantic Intuition

path quantifier

temporal operator

AG p ...along *All* paths p holds *Globally*

EG p ...there *Exists* a path where p holds *Globally*

AF p ...along *All* paths p holds at some state in the *Future*

EF p ...there *Exists* a path where p holds at some state in the *Future*

Computational Tree Logic (CTL)

Syntax

```
Φ ::=  P                                                  ...primitive propositions
    | !Φ  | Φ && Φ | Φ || Φ | Φ -> Φ                     ...propositional connectives
    | AG Φ | EG Φ | AF Φ | EF Φ                           ...path/temporal operators
    | AX Φ | EX Φ | A[Φ U Φ] | E[Φ U Φ]
```

Semantic Intuition

AX p ...along *All* paths, p holds in the *neXt* state

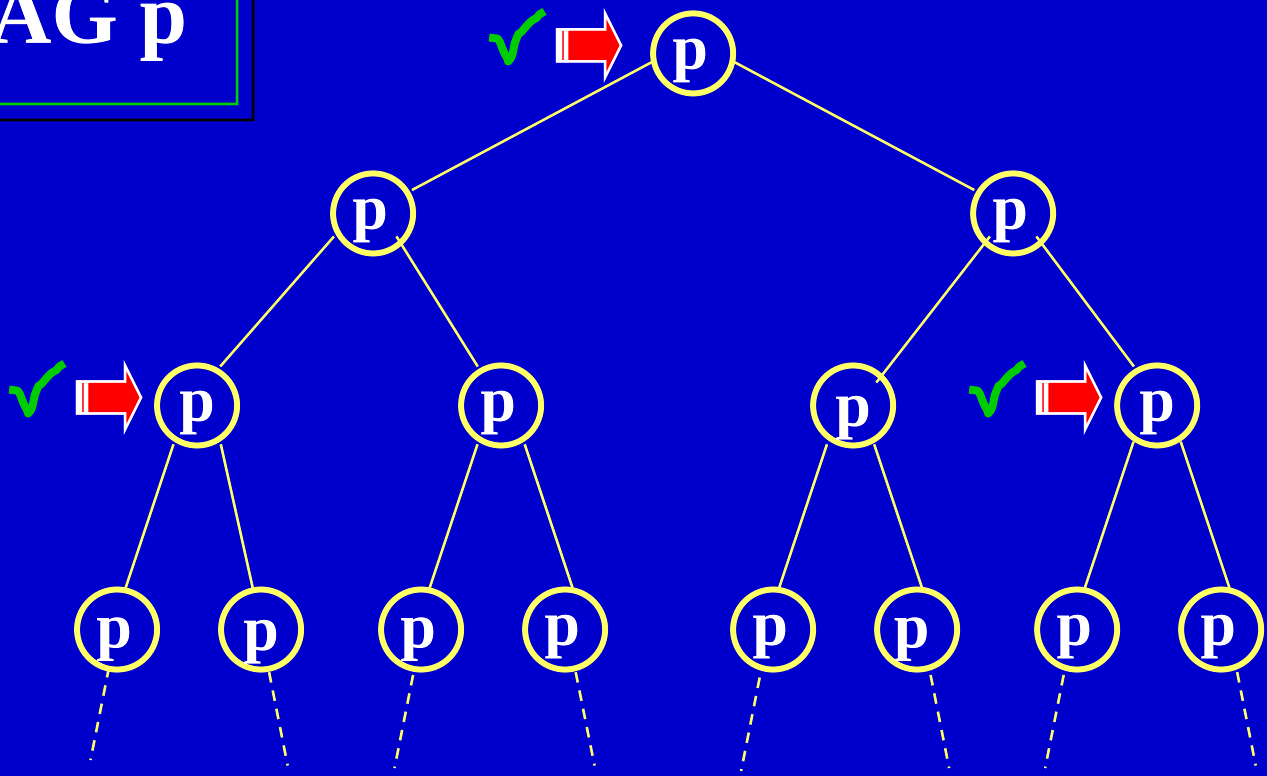
EX p ...there *Exists* a path where p holds in the *neXt* state

A[p U q] ...along *All* paths, p holds *Until* q holds

E[p U q] ...there *Exists* a path where p holds *Until* q holds

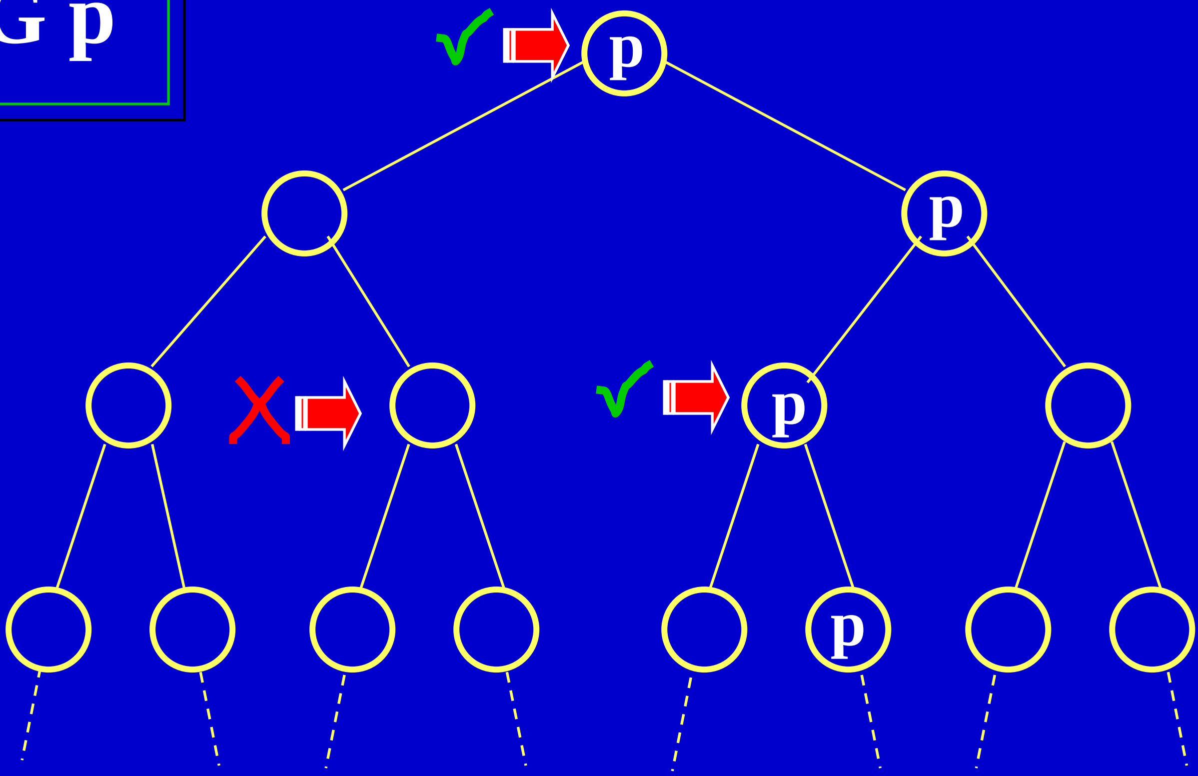
Computation Tree Logic

AG p



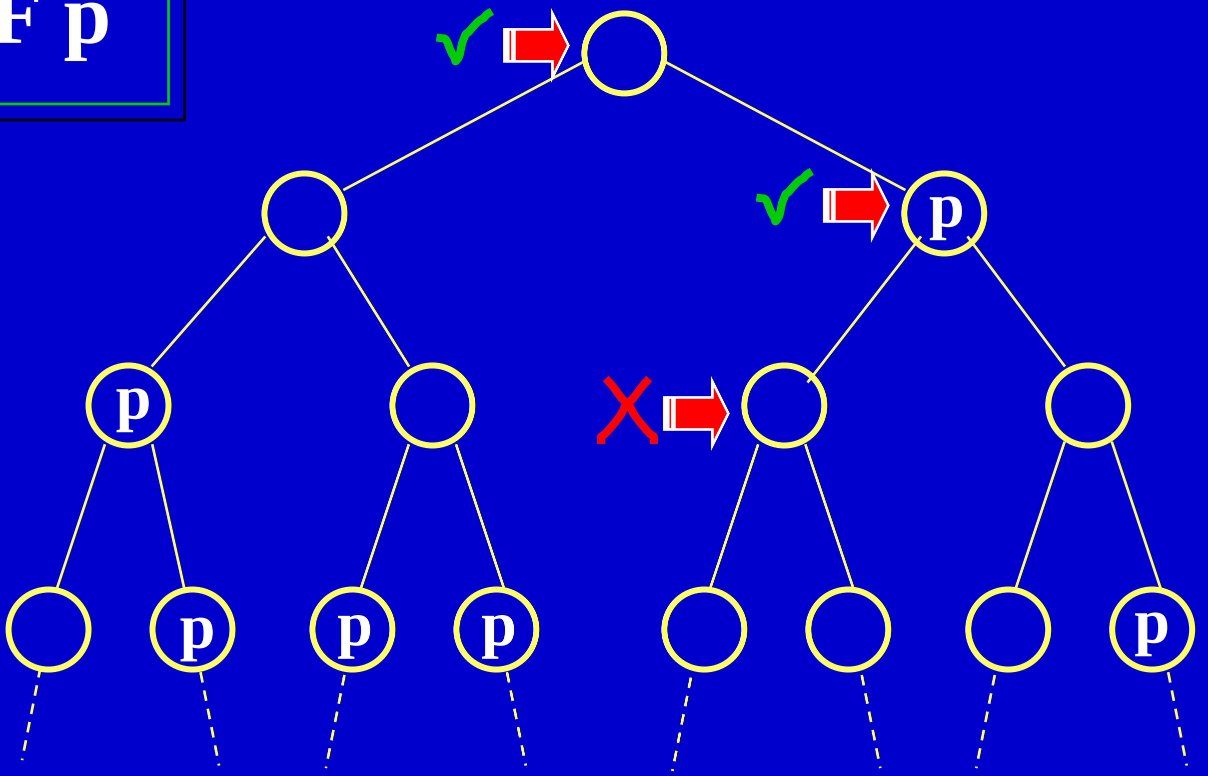
Computation Tree Logic

EG p

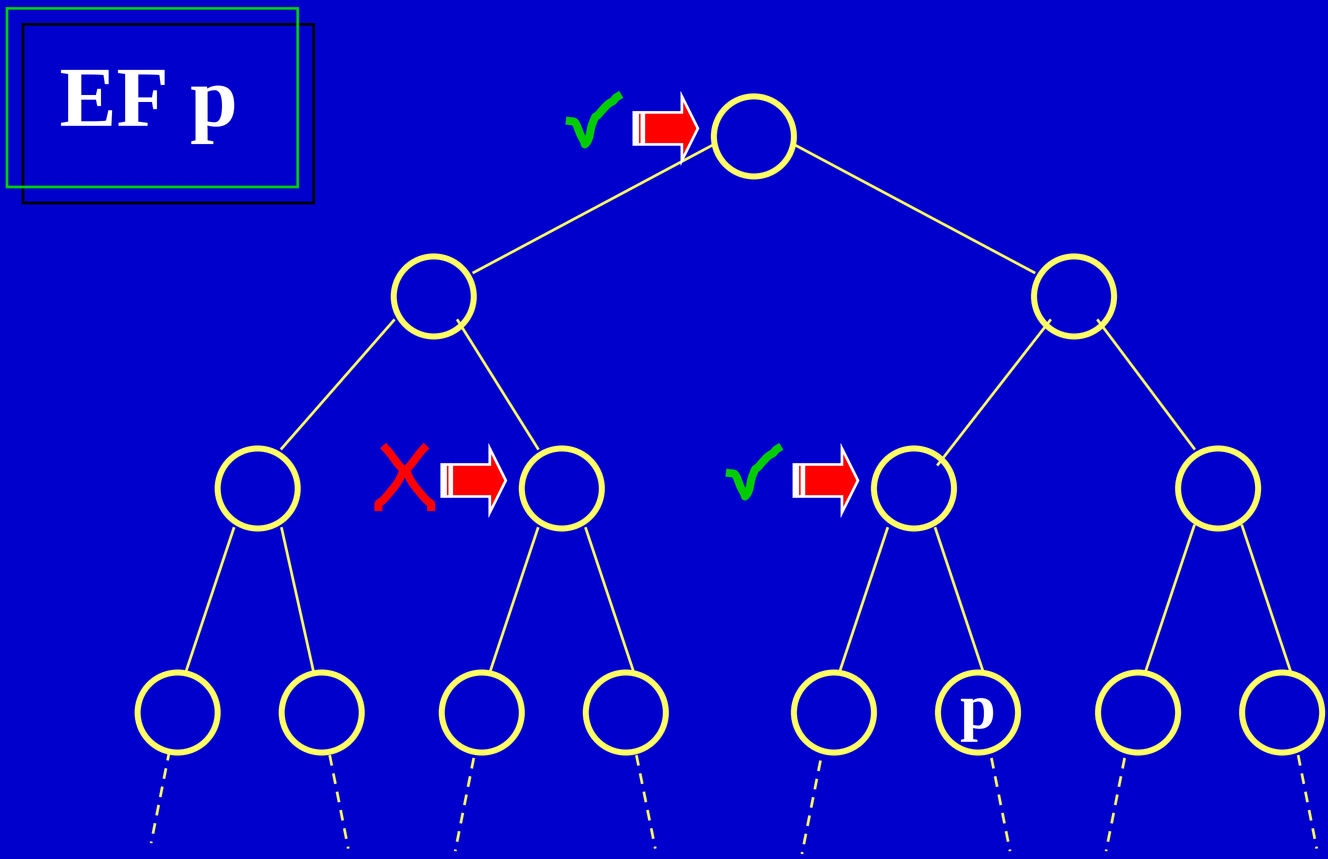


Computation Tree Logic

AF p

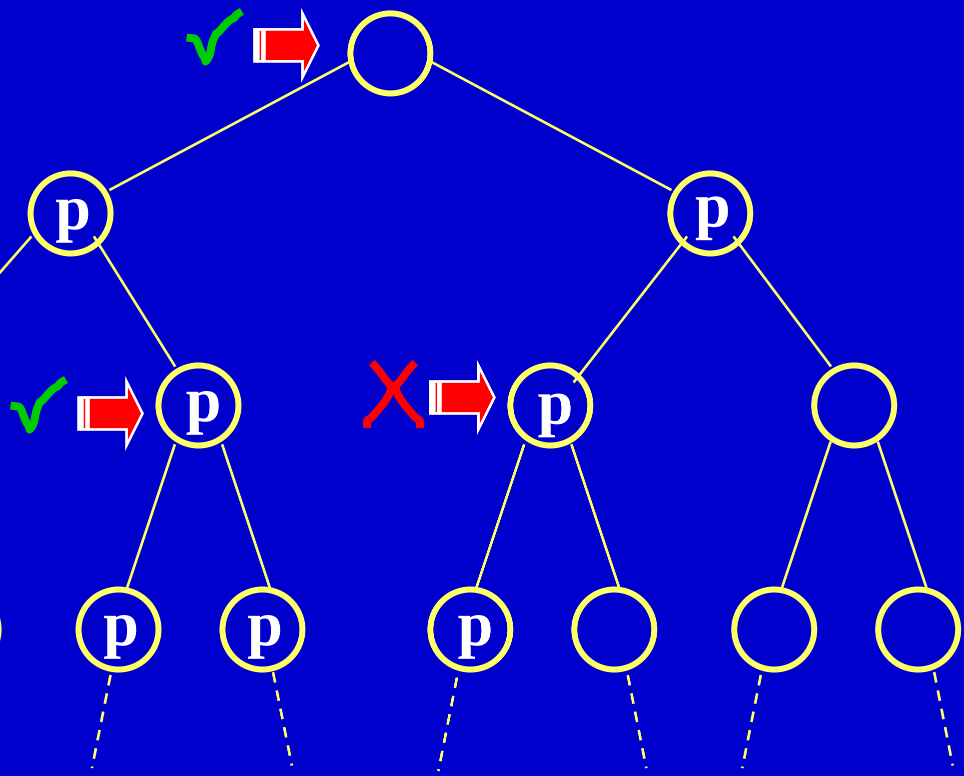


Computation Tree Logic



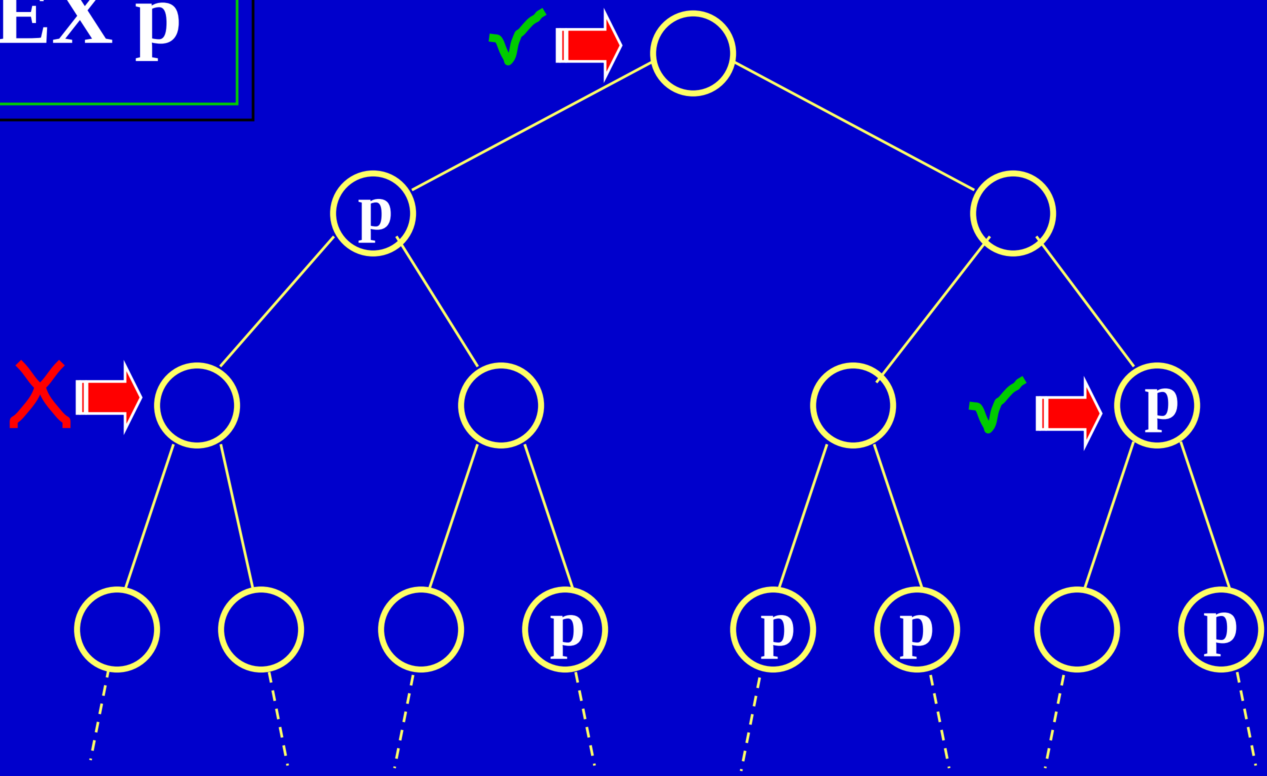
Computation Tree Logic

AX p



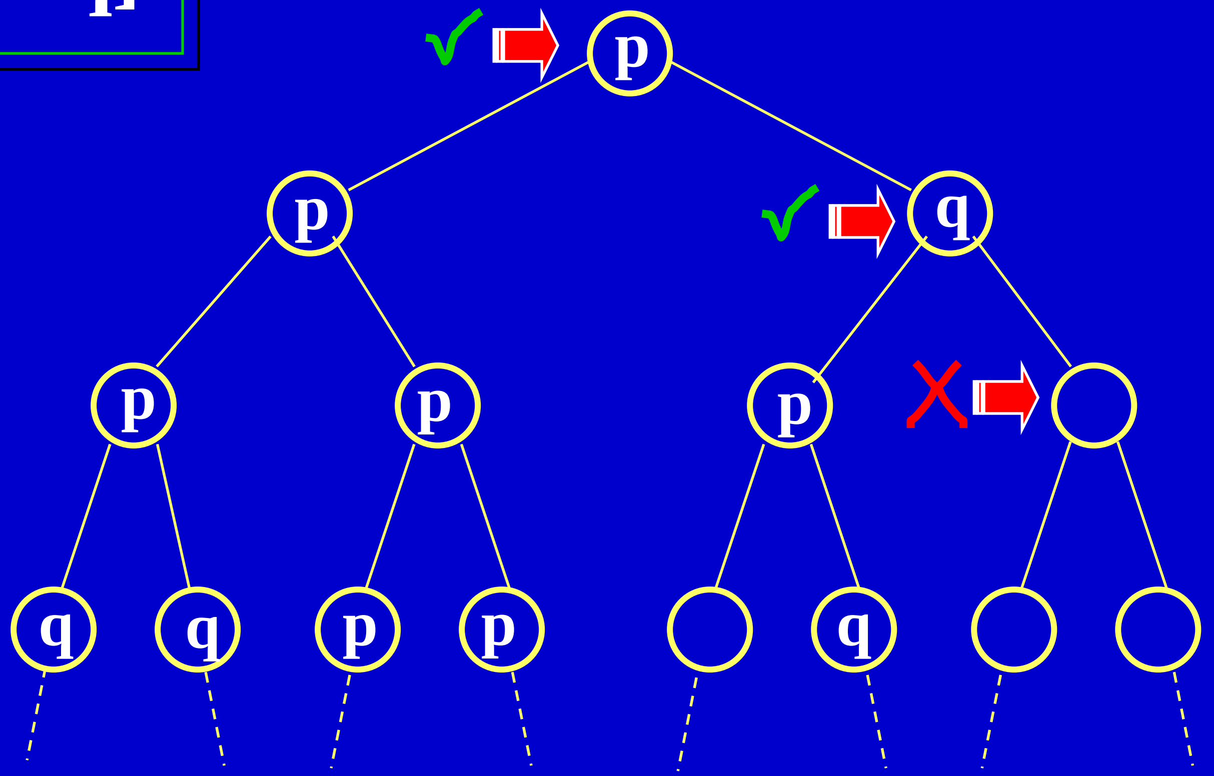
Computation Tree Logic

EX p



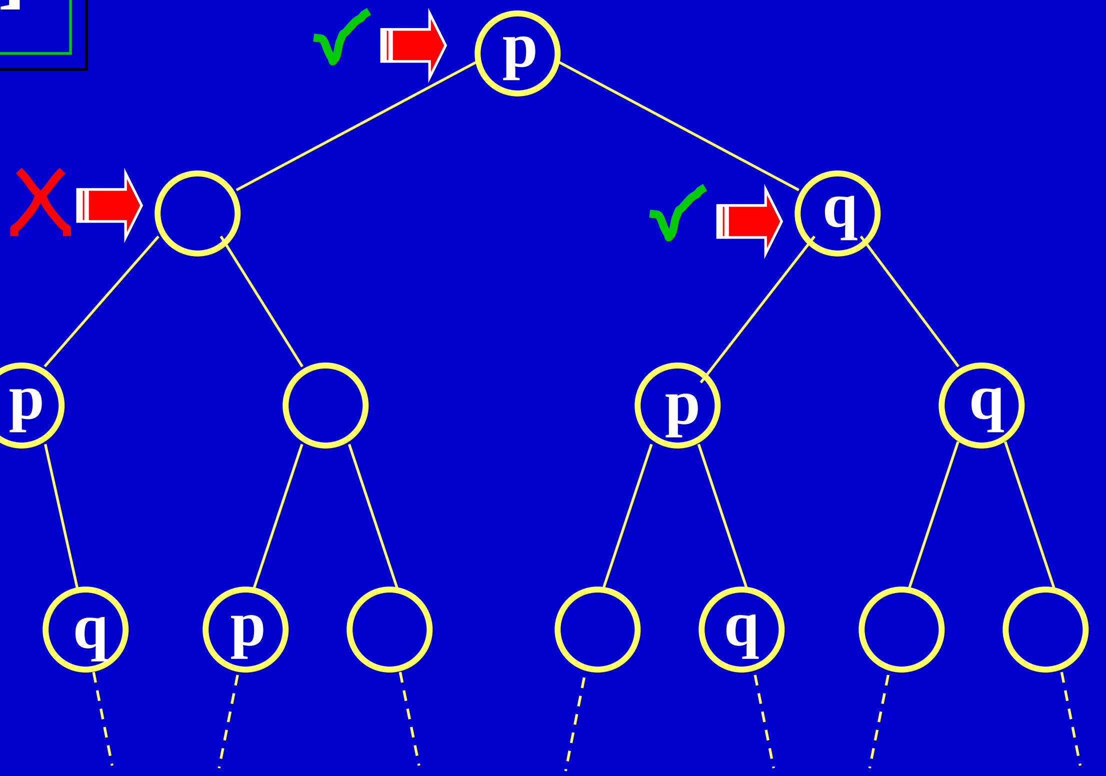
Computation Tree Logic

A[p U q]



Computation Tree Logic

E[p U q]



Example CTL Specifications

- For any state, a request (for some resource) will eventually be acknowledged

AG(requested -> AF acknowledged)

- From any state, it is possible to get to a restart state

AG(EF restart)

- An upwards travelling elevator at the second floor does not changes its direction when it has passengers waiting to go to the fifth floor

AG((floor=2 && direction=up && button5pressed)
-> A[direction=up U floor=5])

Exercices

- **Ecrire en CTL:**
  - **P est vrai après Q**
  - **P devient vrai après Q**
  - **P répond à Q**
  - **On ne peut pas aller plus de 2 fois dans un état vérifiant P**

Exercices --- Corrections ---

- **Ecrire en CTL:**
  - **P est vrai après Q** **AG(Q -> AG(P))**
  - **P devient vrai après Q**
    - **AG (!P U (Q & AF(P)))**
  - **P répond à Q** **AG(Q -> AF(P))**
  - **On ne peut pas aller plus de 2 fois dans un état vérifiant P**
    - **!EF (!P & EX(P & EF(!P & EX(P & EF(!P & EX(P))))))**

Exercice: Minimality

It is sufficient to define CTL syntax as:

```
Φ::=  P
     | !Φ  | Φ && Φ
     | AX Φ | EX Φ
     | A[Φ U Φ] | E[Φ U Φ]
```

Express the other operators as derivatives:

```
f || g =
AF g   =
EF g   =
AG f   =
EG f   =
```

Exercice: Minimality
--- Corrections ---

It is sufficient to define CTL syntax as:

```
Φ  ::=  P
     | !Φ   | Φ && Φ
     | AX Φ | EX Φ
     | A[Φ U Φ] | E[Φ U Φ]
```

Express the other operators as derivatives:

```
f || g =   ! (!f && !g)
AF g   =   A[true U g]
EF g   =   E[true U g]
AG f   =   ! E[true U !f]
EG f   =   ! A[true U !f]
```

Semantics: interpretation on Kripke structures

- Kripke structure K = (S,R,L)
  - S set of states
  - R transition relation
  - L valuation function L(ρ)(s) -> True/False
- Path = infinite sequence $(s_0, s_1, s_2, \dots)$
  such that $\forall i\ (s_i, s_{i+1}) \in R$

Semantics: interpretation on Kripke structures

Formalisation of the semantics :

$s \models p$ iff $L(s)(p)$ where p atomic proposition

$s \models !f$ iff $s \not\models f$

$s_0 \models AX\, f$ iff for all paths $(s_0, s_1, s_2, \ldots)$, $s_1 \models f$

$s_0 \models A(f\, U\, g)$ iff for all paths $(s_0, s_1, \ldots)$, for some i, $si \models f$ and for all j<i $sj \models g$

Exercice:

$s0 \models AG\, f$ iff

$s0 \models EF\, f$ iff

Interpretation on Kripke structures --- Corrections ---

Formalisation of the semantics :

$s \models p$ iff $L(s)(p)$ where p atomic proposition

$s \models !f$ iff $s \not\models f$

$s_0 \models AX\ f$ iff for all paths $(s_0, s_1, s_2, \ldots)$, $s_1 \models f$

$s_0 \models A(f\ U\ g)$ iff for all paths $(s_0, s_1, \ldots)$, for some i, $si \models f$ and for all $j<i$ $sj \models g$

Exercice:

$s0 \models AG\ f$ iff for all paths $(s0,s1, s2, \ldots)$, for all i, $si \models f$

$s0 \models EF\ f$ iff there exists a path $(s0,s1, s2, \ldots)$, and an i, with $si \models f$

Modal Logics

Temporal logics for Labelled Transition Systems (= action-based)

- **HML (Hennessy-Milner, 85)**
- **ACTL (DeNicola-Vandrager, 90)**
- **Modal μ-calculus (Kozen 83)**
- **Regular μ-calculus (Madescu 03)**

ACTL: Action Computation Tree Logic

- Atomic propositions (on actions) + boolean connectors
- Paths formulas:

Next

$$\psi ::= X_\alpha \varphi \qquad [\![X_\alpha\varphi]\!] = \{s_1 \xrightarrow{a_1} s_2 \cdots \mid a_1 \in [\alpha] \wedge s_2 \in [\![\varphi]\!]\}$$

$$\mid \; X_\tau \varphi \qquad [\![X_\tau\varphi]\!] = \{s_1 \xrightarrow{\tau} s_2 \cdots \mid s_2 \in [\![\varphi]\!]\}$$

ACTL:
Action Computation Tree Logic

- Paths formulas:

Until

$$
\begin{array}{lll}
\mid & \varphi_1{}_\alpha\mathsf{U}\ \varphi_2 & [\![\varphi_1{}_\alpha\mathsf{U}\ \varphi_2]\!] = \{s_1 \xrightarrow{a_1} \cdots \xrightarrow{a_{i-1}} s_i \cdots \mid i \geq 1 \wedge s_i \in [\![\varphi_2]\!] \wedge \\
& & \quad \forall j \in [1, i-1].a_j \in [\![\alpha \vee \tau]\!] \wedge s_j \in [\![\varphi_1]\!]\} \\
\mid & \varphi_1{}_{\alpha_1}\mathsf{U}_{\alpha_2}\varphi_2 & [\![\varphi_1{}_{\alpha_1}\mathsf{U}_{\alpha_2}\varphi_2]\!] = \{s_1 \xrightarrow{a_1} \cdots \xrightarrow{a_{i-1}} s_i \cdots \mid i \geq 2 \wedge s_i \in [\![\varphi_2]\!] \wedge \\
& & \quad a_{i-1} \in [\![\alpha_2]\!] \wedge s_{i-1} \in [\![\varphi_1]\!] \wedge \\
& & \quad \forall j \in [1, i-2].a_j \in [\![\alpha_1 \vee \tau]\!] \wedge s_j \in [\![\varphi_1]\!]\}
\end{array}
$$

ACTL:
Action Computation Tree Logic

- State formulas:

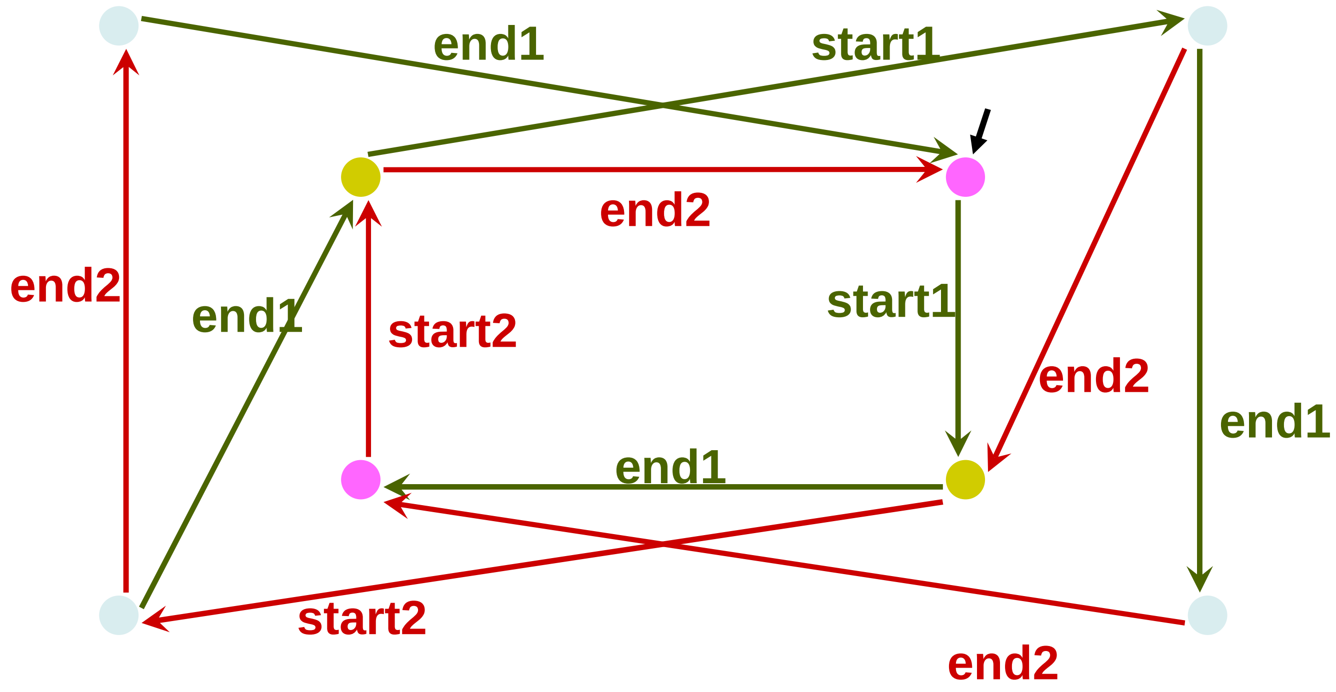
$$\varphi ::= \mathsf{ff} \qquad\qquad [\![\mathsf{ff}]\!] = \emptyset$$
$$\mid\ \mathsf{E}\psi \qquad\qquad [\![\mathsf{E}\psi]\!] = \{s \in S \mid \exists p \in Path(s).p \in [\![\psi]\!]\}$$
$$\mid\ \mathsf{A}\psi \qquad\qquad [\![\mathsf{A}\psi]\!] = \{s \in S \mid \forall p \in Path(s).p \in [\![\psi]\!]\}$$

Note the recursive def of path/state formulas.

Define derived operators as usual:

$$\mathsf{EF}_\alpha\varphi = \mathsf{E}(\mathsf{tt}_\alpha \mathsf{U}\ \varphi)\ \text{ et }\ \mathsf{AG}_\alpha\varphi = \neg\mathsf{EF}_\alpha\neg\varphi.$$
$$\langle\alpha\rangle\,\varphi\ =\ \mathsf{EX}_\alpha\varphi\ \text{ et }\ [\alpha]\,\varphi\ =\ \neg\,\langle\alpha\rangle\,\neg\varphi$$

Exemple: Scheduler_2

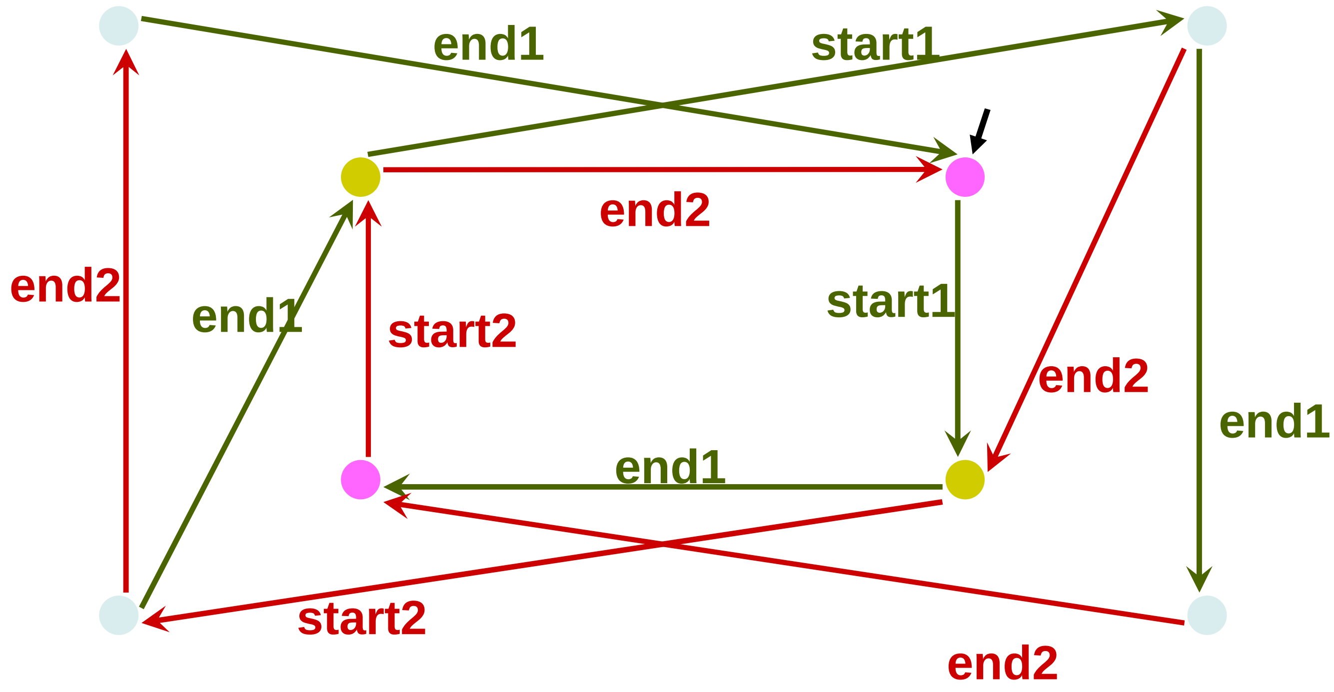


i,j in {1,0} i≠j :

AG_{tt} [start_i] $AG_{!end\_i}$ [start_j] ff

Or equivalently : $!EF_{tt}$ [start_i] $EF_{!end\_i}$ [start_j] tt

Exemple: Scheduler_2

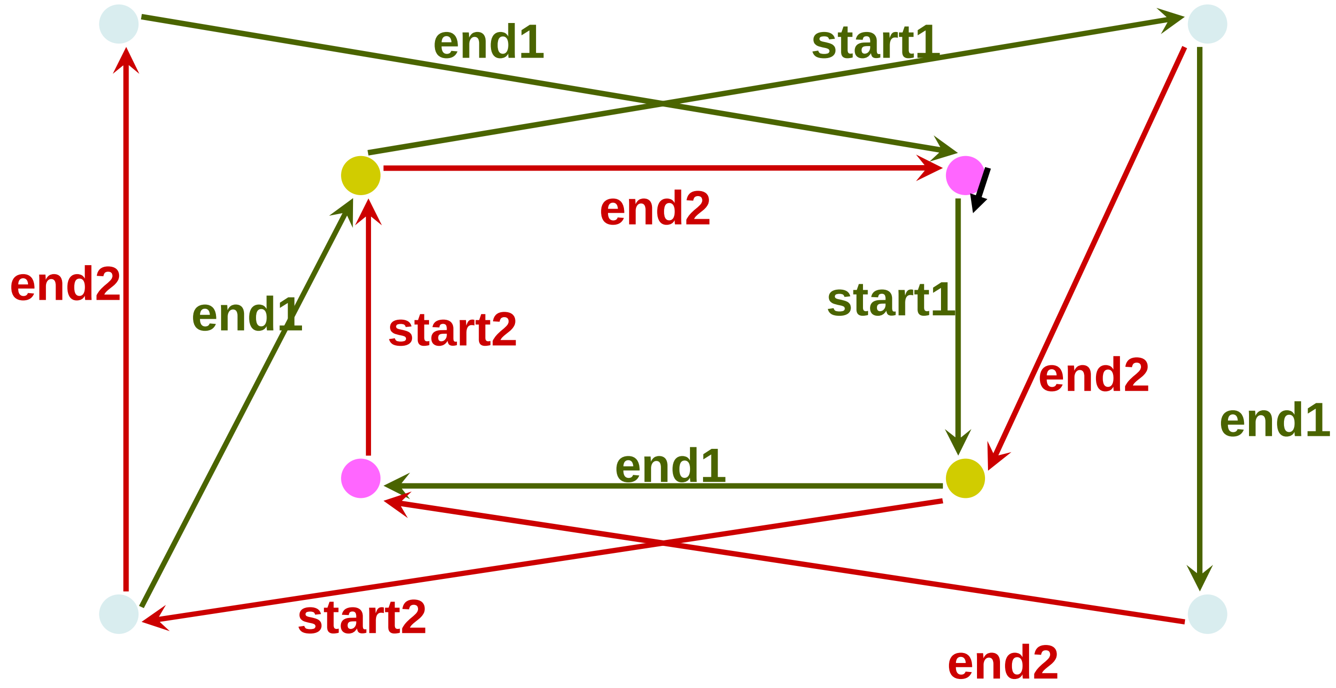


Que signifie ?

$$AG_{tt}\ (EF_{tt}\ \langle end\_i \rangle\ tt \wedge\ EF_{tt}\ \langle start\_i \rangle\ tt)$$

--- Corrections ---

Exemple: Scheduler_2



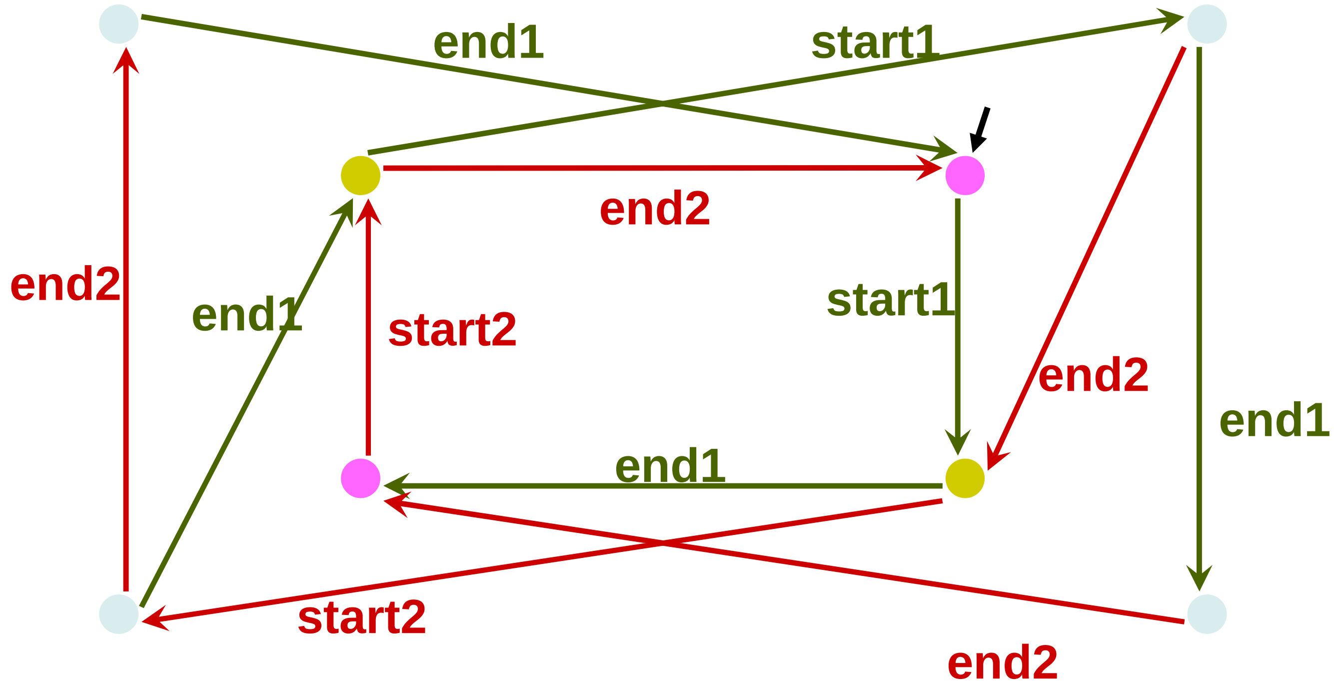
Que signifie ?

$AG_{tt}\ (EF_{tt}\ \langle end\_i \rangle\ tt \wedge EF_{tt}\ \langle start\_i \rangle\ tt)$

Vivacité :

ttes les actions visibles sont toujours atteignables

Exemple: Scheduler_2

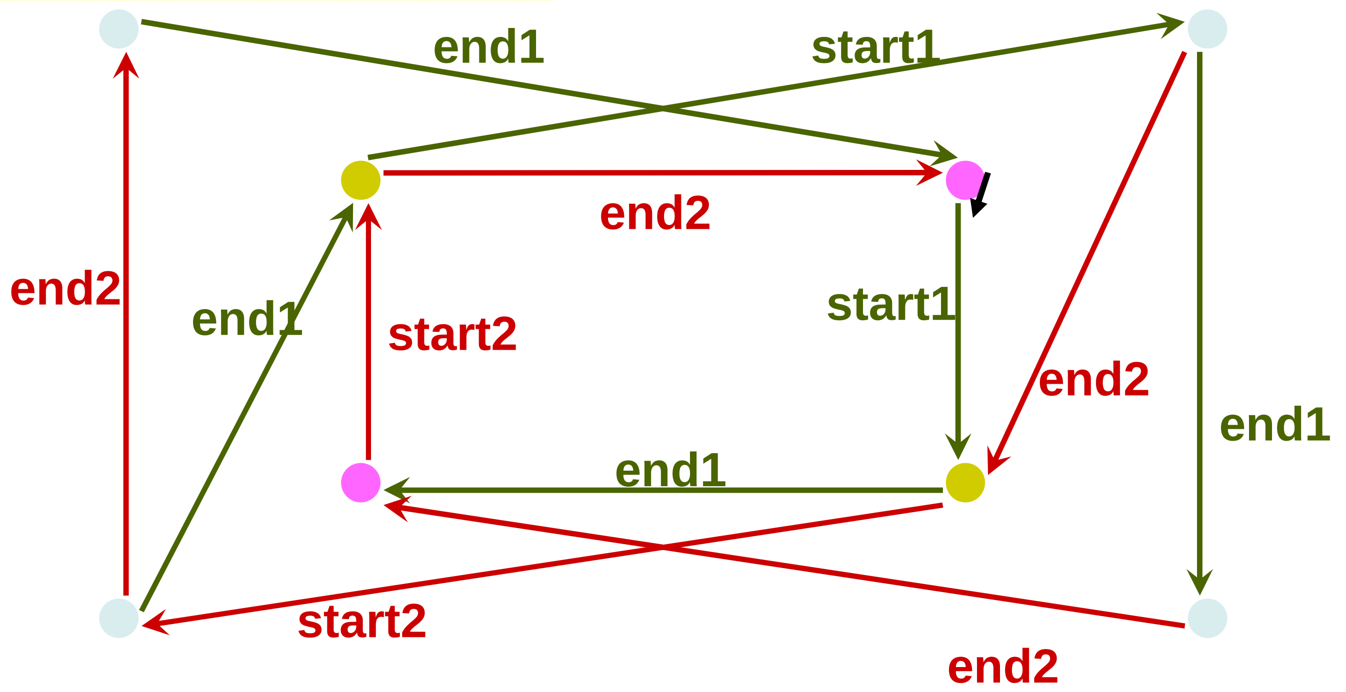


Que signifie ?

$$AG_{tt}\,[end\_i]\; A\,(tt\;_{tt}U_{start\_i}\; tt)$$

--- Corrections ---

Exemple: Scheduler_2



Que signifie ?

$AG_{tt}\,[end\_i]\ A\ (tt\ {}_{tt}U_{start\_i}\ tt)$

Inévitabilité / absence de famine :

pour chaque i, start_i est inévitable en un nombre fini de transition à partir de n'importe quel end_i

Temporal Logics

- Temporal Logic : CTL
- Modal logic: ACTL
- Logic patterns

Motivation for Specification Patterns

- Temporal properties are not always easy to write
- Clearly many specifications can be captured in both CTL and ACTL (or LTL*)

* left for personal research

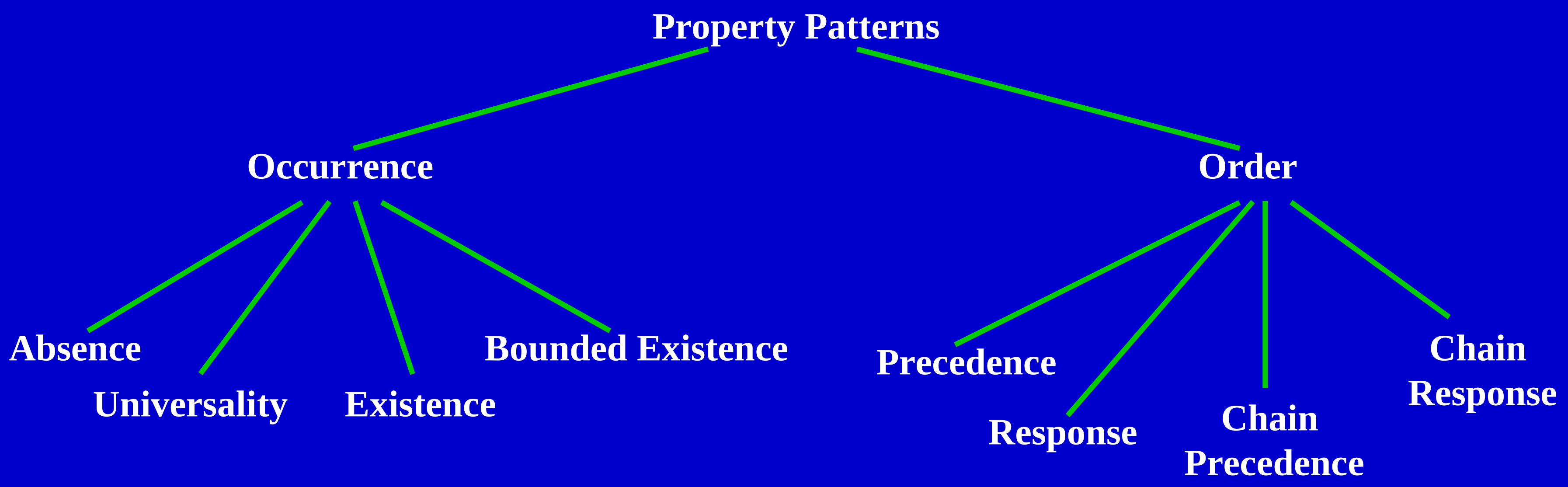
Example: **action Q must respond to action P**

CTL: `AG(P -> AF Q)` **LTL:** `[](P -> <>Q)`

You can use specification patterns to:

- Capture the experience base of expert designers
- Transfer that experience between practitioners.

Pattern Hierarchy



Classification

- *Occurrence* Patterns:
  – require states/events to occur or not to occur
- *Order* Patterns
  – constrain the order of states/events

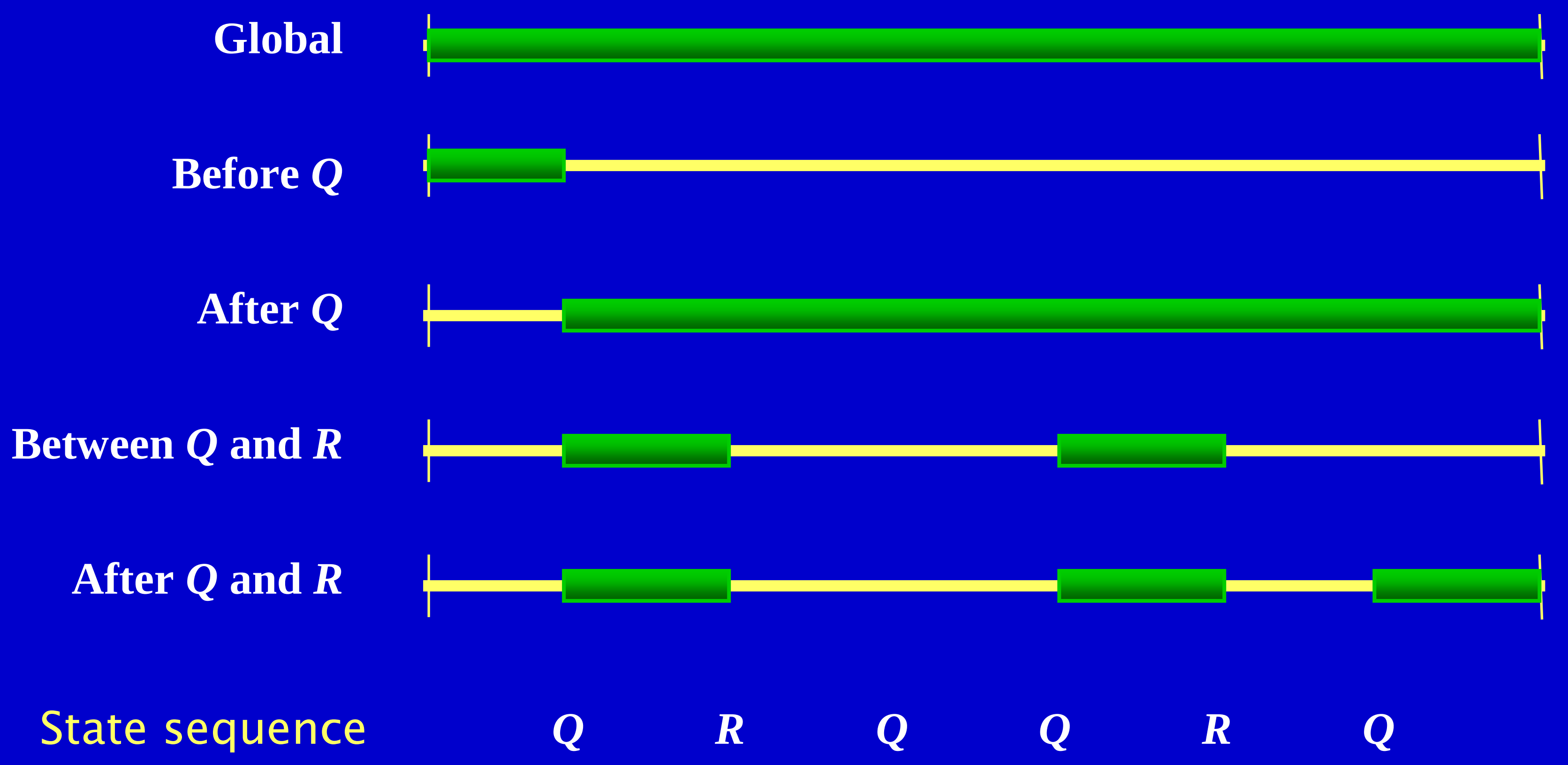
Occurrence Patterns

- Absence: A given state/event does not occur within a scope
- Existence: A given state/event must occur within a scope
- Bounded Existence: A given state/event must occur *k* times within a scope
  - variants: *at least k* times in scope, *at most k* times in scope
- Universality: A given state/event must occur throughout a scope

Order Patterns

- Precedence: A state/event P must always be preceded by a state/event Q within a scope
- Response: A state/event P must always be followed a state/event Q within a scope
- Chain Precedence: A sequence of state/events P1, …, Pn must always be preceded by a sequence of states/events Q1, …, Qm within a scope
- Chain Response: A sequence of state/events P1, …, Pn must always be followed by a sequence of states/events Q1, …, Qm within a scope

Pattern Scopes



The Response Pattern

Intent

To describe cause-effect relationships between a pair of events/states. An occurrence of the first, the cause, must be followed by an occurrence of the second, the effect. Also known as **Follows** and **Leads-to**.

Mappings: *In these mappings, **P** is the cause and **S** is the effect*

LTL:

Globally:	`[](P -> <>S)`
Before R:	`<>R -> (P -> (!R U (S & !R))) U R`
After Q:	`[](Q -> [](P -> <>S))`
Between Q and R:	`[]((Q & !R & <>R) -> (P -> (!R U (S & !R))) U R)`
After Q until R:	`[](Q & !R -> ((P -> (!R U (S & !R))) W R)`

The Response Pattern (continued)

Mappings: *In these mappings, **P** is the cause and **S** is the effect*

CTL:

Globally: `AG(P -> AF(S))`

Before R: `A[((P -> A[!R U (S & !R)]) | AG(!R)) W R]`

After Q: `A[!Q W (Q & AG(P -> AF(S))]`

Between Q and R: `AG(Q & !R -> A[((P -> A[!R U (S & !R)]) | AG(!R)) W R])`

After Q until R: `AG(Q & !R -> A[(P -> A[!R U (S & !R)]) W R])`

Examples and Known Uses:

Response properties occur quite commonly in specifications of concurrent systems. Perhaps the most common example is in describing a requirement that a resource must be granted after it is requested.

Relationships

Note that a Response property is like a converse of a Precedence property. Precedence says that some cause precedes each effect, and...

Specify Patterns in Bandera

The Bandera Pattern Library is populated by writing pattern macros:

```
pattern {
    name = “Response”
    scope = “Globally”
    parameters = {P, S}
    format = “{P} leads to {S} globally”
    ltl = “[]({P} -> <>{S})”
    ctl = “AG({P} -> AF({S}))”
}
```

Evaluation (Kansas University,)

- 555 TL specs collected from at least 35 different sources
- 511 (92%) matched one of the patterns
- Of the matches...
  - Response: 245 (48%)
  - Universality: 119 (23%)
  - Absence: 85 (17%)

Questions

- Do patterns facilitate the learning of specification formalisms like CTL and LTL?
- Do patterns allow specifications to be written more quickly?
- Are the specifications generated from patterns more likely to be correct?
- Does the use of the pattern system lead people to write more expressive specifications?

Based on anecdotal evidence, we believe the answer to each of these questions is “yes”

Beyond LTL/CTL/ACTL: Logics with data

MCL : Model Checking Language (Matescu 2008)

= regular modal μ-calculus + data

$$
\begin{array}{l}
[\ \mathsf{true}^*.\{\mathit{cmd}\ ?i{:}\mathsf{nat}\ \mathsf{where}\ i < n_c\}\ ]\\
\quad \mathsf{forall}\ j{:}\mathsf{nat}\ \mathsf{among}\ \{i+1\ \ldots\ n-1\}.\\
\qquad (j \neq n_c) \Rightarrow \langle(\neg\{\mathit{rec}\ !i\})^*.\{\mathit{cmd}\ !j\}.(\neg\{\mathit{rec}\ !i\})^*.\{\mathit{rec}\ !j\}\rangle\ @
\end{array}
$$

1: receive a value (with a condition)

2: data quantification

3: regular expressions, modalities, infinite loops, etc.

(reduces the need for writing explicit fix-points)

Vocabulary: back on important notions

- Safety / Liveness
- What does it means
- What kind of diagnostics ?

Safety Properties

- Informally, a safety property states that
  nothing bad ever happens
- Examples
  - Invariants: “x is always less than 10”
  - Deadlock freedom: “the system never reaches a state where no moves are possible”
  - Mutual exclusion: “the system never reaches a state where two processes are in the critical section”
- As soon as you see the “bad thing”, you know the property is false
- Safety properties can be falsified by a finite-prefix of an execution trace
  - Practically speaking, an error trace for a safety property is a finite list of states beginning with the initial state

Liveness Properties

- Informally, a liveness property states that
  something good will eventually happen
- Examples
  - Termination: "the system eventually terminates"
  - Response properties: "if action X occurs then eventually action Y will occur"
- Need to keep looking for the "good thing" forever
- Liveness properties can be falsified by an infinite-suffix of an execution trace
  - Practically speaking, an error trace for a liveness property is a finite list of states beginning with the initial state followed by a *cycle* showing you a loop that can cause you to get stuck and never reach the "good thing"

Safety vs Liveness

- **Practically, it is important to know the difference because…**
  - **It impacts how we design verification algorithms and tools**
    - **Some tools only check safety properties (e.g., based on *reachability* algorithms)**
  - **It impacts how we run tools**
    - **Different command line options are used for Spin**
  - **It impacts how we form abstractions**
    - **Liveness properties often require forms of abstraction that differ from those used in safety properties**

Assessment

- Safety vs Liveness is an important distinction
- However, it is very coarse
  - Lots of variations within safety and liveness
  - A finer classification might be more useful
- Liveness is more useful when used with "fairness" conditions.

Summary

- Computational Tree Logic : CTL
  - Properties of executions in non-deterministic state-based models
- Modal logic: ACTL
  - Idem, for action-based models
- Logic patterns
  - User friendly / natural language like constructs
  - With a formal definition !